15-440/640: Project 4
Clustering Data Points and DNA Strands Using MPI

Project posted on: Thursday, November 14, 2013
Due date: Thursday, December 5, 2013 at 11:59:59 PM

Project Objectives

This project has three goals:

1. To develop, by doing, a clear understating on how to parallelize an algorithm for distributed processing using OpenMPI for communication.
2. To gain a better understanding of the effects of scale on the performance of distributed processing by conducting and analyzing scalability studies on various degrees of parallelism and data set sizes.
3. To obtain exposure to, and an appreciation for, a common real-world clustering problem , because cluster analysis represents an important class of problem in various domains including, but not limited to, data mining and statistical data analysis.

Cluster Analysis

Cluster analysis or **clustering** is the task of assigning a set of objects into groups (called **clusters**) so that the degree of similarity can be strong between members of the same cluster and weak between members of different clusters. In summary, clustering has to define some notion of "similarity" among objects. The objective is to maximize intra-cluster similarity and minimize inter-cluster similarity. In this project we will apply K-Means clustering to two different applications, data points in a 2D plane and DNA strands in biology.
Clustering problems arise in many different applications such as visualization (e.g., visualizing the stock market data to give individuals/institutions useful information about the market behavior for investment decisions), data mining and statistical data analysis including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Among clustering formulations that are based on minimizing a formal objective function, perhaps the most widely used and studied one is K-Means algorithm. Simply put, K-Means is an iterative algorithm that attempts to find K similar groups in a given data set via minimizing a mean squared distance function. Initial guesses of K means (**m1**, **m2**, …, **mK**) is firstly made (see Fig. 1(a)). These estimated means are then used to classify the data set objects into K clusters. Afterwards, each mean is recomputed so as to reflect the true mean of its constituent objects (see Fig. 1(b)). The algorithm keeps iterating until the recomputed means (almost) stop varying (see Fig. 1(c)).

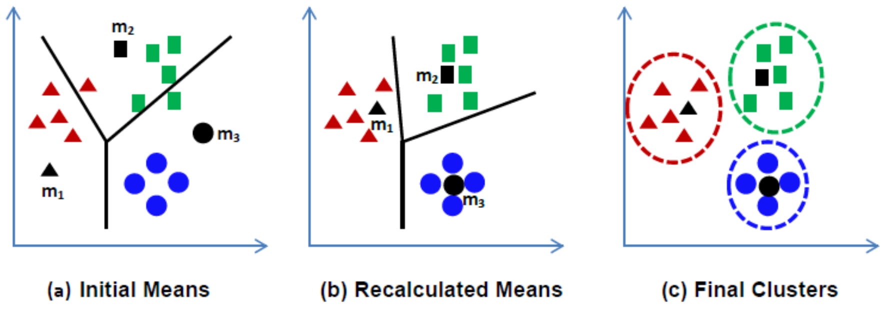


Fig. 1

Clustering Data Points

Considering a case of a data set composed of data points in ***d-dimensional*** space ***Rd***. In K-Means clustering, we specify a set of ***n*** data points and an integer ***k***. The problem then is to determine a set of ***k*** points in ***Rd***, called ***centroids***, so as to minimize the ***mean squared distance*** from each data point to its nearest center. In pseudo code, it is shown by Alpaydin (Introduction to Machine Learning, page 139) that K-Means essentially follows the following procedure:

```
Initialize $m_i$ to k random $x^d$, for i = 1, …., k and $x^d \in X$ that contains
each of our d-dimensional data point.
Repeat
For all $x^d$ in X
   bid ← 1 if $||x^d - m_i|| = \min_j ||x^d - m_j||$
   bid ← 0 otherwise
For all $m_i$, i = 1, …., k
  M ← sum over $b_i^d x^d$/sum over $b_i^d$
Until $m_i$ converge
```

Explained in plain English, K-Means roughly follows this approach:

1. We start by deciding how many clusters we would like to form from our data. We call this value ***k***. The value of ***k*** is generally a small integer, such as 2, 3, 4, or 5, but may be larger.

2. Next we select ***k*** points to be the centroids of ***k*** clusters which at present have no members. The list of centroids can be selected by any method (e.g., randomly from the set of data points). It is usually better to pick centroids that are far apart.

3. We then compute the ***Euclidean distance*** (*the similarity function with a data set of data points*) from each data point to each centroid. A data point is assigned to a cluster such that its distance to that cluster is the smallest among all other distances.

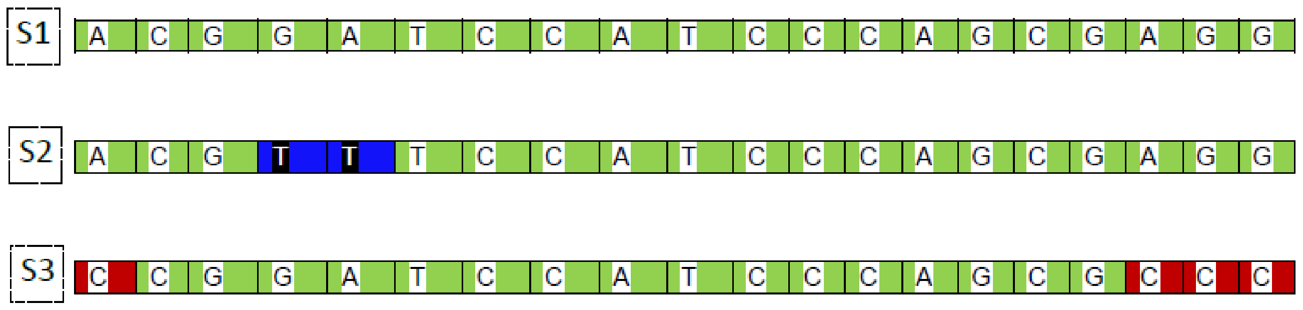
4. After associating every data point with one of ***k*** clusters, each centroid is recalculated so as to reflect the true *mean* of its constituent data points.

5. Steps 3 and 4 are repeated for a number of times (say **μ**), essentially until the centroids start varying very little.

The positive integer **μ** is known as number of K-Means iterations. The precise value of **μ** can vary depending on the initial starting cluster centroids, even on the same data set.

Clustering DNA Strands

Bioinformatics involves the manipulation, searching, and data mining of biological data, and this includes DNA sequence data. A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine (A), guanine (G), cytosine (C), and thymine (T). We can express a strand of DNA as a string over the finite set **{**A, C, G, T**}**. String searching or matching algorithms, which find an occurrence of a sequence of letters inside a larger sequence of letters, or simply match two sequences of letters, is widely used in genetics (e.g., for studying various phylogenetic relationships and protein functions). In many of studies, we often want to compare the DNA of two (or more) different organisms. One goal of comparing two strands of DNA is to determine how "similar" the two strands are, as some measure of how closely related the two organisms are. ***Similarity in such a scenario can be defined as a function F(. , .) of the number bases in a strand subtracted from the number of changes required to turn one strand into the other***. For example consider the following three DNA strands:


S1
A C G G A T C C A T C C C A G C G A G G
S2
A C G T T T C C A T C C C A G C G A G G
S3
C C G G A T C C A T C C C A G C G C C C

The similarity between S1 and S2 is denoted as F(S1, S2) and is equal to 18. On the other hand, F(S1, S3) = 16. The K-Means algorithm, described in the previous section, can be applied to DNA strands with this given similarity function F(. , .) to compare DNA of two or more different organisms.

Implementation Requirements

You should implement ***sequential and parallel versions*** of K-Means with two types of data sets, one consisting of 2D data points and the other of DNA strands. For simplicity assume that strands in the DNA data set are equal in size. In addition, you have to write your own data set generator for each data type that generates a given number of data points or DNA strands. You can use any programming language of your choice.

Your sequential and MPI-based K-Means implementations should be tested and run on a data set of 2D data points, generated by the given data set generator, and on a data set of DNA strands, generated by your DNA strands data generator.

Experimentation and Analysis

Please evaluate the performance, specifically total time across all processors and start-to-finish time for the system as a whole, of your MPI-based solution as you vary the number of processes with a fixed data set of 2D data points. Specifically, consider at least 2, 4, 8, and 12 processes. How does the performance scale? Is there a sweet spot? A point of diminishing marginal returns? Illustrate your assessment with one or more graphs, if possible.

Deliverables

1. Your fully tested and debugged code for your data set generator, and your sequential and MPI-based solutions.
2. A report, in .pdf format, that (i) describes your approach to parallelizing the algorithm, including pseudocode, and (ii) the findings from your "Experimentation and Analysis", as described above.

OpenMPI Cluster

OpenMPI is installed in all of the Gates cluster machines. In general, these are machines *ghcXX.ghc.andrew.cmu.edu*, where *XX* ranges from 01, 02, 03, 04, 05, …, 81.

In order to make use of the cluster,

(1) first log in, locally or via *ssh*, to one of the machines. From there, *ssh* to several other machines and ensure that your ~/.ssh/known_hosts file is up-to-date. For OpenMPI to work, you need to be able to *ssh* to the machines you will be using without entering a password.

(2) Copy the list of machines names you have chosen and tested into a text file, one machine name per line. You will feed this to OpenMPI to let it know what machines you want to use. There are no specific requirements in naming the file.

(3) OpenMPI is installed in */usr/lib64/openmpi*. The *bin* and *lib* directories within it need to be made part of your environment – both locally and remotely. Specifically, the LD_LIBRARY_PATH and PATH environment variables need to be set automatically, without human involvement, upon logging in. For example, if your login shell is *csh*, you probably want to add these lines to your *.cshrc* file:

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/local/lib/openmpi/lib
setenv PATH ${PATH}:/usr/local/lib/openmpi/bin
```

Resources

Check the resources link on the Web for links to the generator, tutorials, etc.